



National Defence
Research and
Development Branch

Défense Nationale
Bureau de Recherche
et Développement

TECHNICAL MEMORANDUM 84/D

March 1984

GETWRD PACKAGE

David Hally • C. Ann Dent

**Defence
Research
Establishment
Atlantic**



**Centre de
Recherches pour la
Défense
Atlantique**

Canada

84- 3556 /

DEFENCE RESEARCH ESTABLISHMENT ATLANTIC

9 GROVE STREET

P.O. BOX 1012
DARTMOUTH, N.S.
B2Y 3Z7

TELEPHONE
(902) 426-3100

CENTRE DE RECHERCHES POUR LA DÉFENSE ATLANTIQUE

9 GROVE STREET

C.P. 1012
DARTMOUTH, N.E.
B2Y 3Z7



National Defence
Research and
Development Branch

Défense Nationale
Bureau de Recherche
et Développement

GETWRD PACKAGE

David Hally • C. Ann Dent

March 1984

Approved by T. Garrett

Director / Technology Division

DISTRIBUTION APPROVED BY

CHIEF D. R. E. A.

TECHNICAL MEMORANDUM 84/D

Defence
Research
Establishment
Atlantic



Centre de
Recherches pour la
Défense
Atlantique

Canada

GETWRD Package

Abstract

The GETWRD Package is a library of twelve procedures designed to ease the implementation of command languages in FORTRAN programs. Use of the GETWRD Package saves the implementor of the program from designing and coding new input routines. The GETWRD Package is able to interpret a *word* of user input by matching it with one of the entries in a user-supplied dictionary. Alternatively the *word* may be interpreted as a number.

The GETWRD Package includes many features designed to increase the friendliness of the program/user interface. These include a type ahead facility, recognition of abbreviations, word completion, and understandable error messages. Also, it provides for individual tailoring of prompts and help messages and, where possible, help messages include the list of allowed responses. A procedure which checks the dictionary for errors is also supplied as a debugging tool.

In this memorandum a complete description of each procedure is given, including the algorithm used, discussion of their use and examples. The GETWRD Package is written in ANSI standard FORTRAN of 1977. Care has been taken to make the package as portable as possible.

RESUME

Le progiciel GETWRD est une bibliothèque de douze procédures conçues pour faciliter l'implantation des ordres de gestion dans les programmes FORTRAN. Grâce à ce progiciel, le responsable de l'implantation du programme n'a plus à concevoir et à écrire des routines d'entrée. Le progiciel GETWRD est capable d'interpréter un mot de l'entrée utilisateur en l'appariant avec une des entrées du dictionnaire fourni par l'utilisateur. Le mot peut aussi être un nombre.

Le progiciel GETWRD comporte de nombreuses fonctions qui facilitent son utilisation, notamment une fonction d'entrée par anticipation, la reconnaissance des abréviations, leur conversion en mots et des messages d'erreur compréhensibles. Il offre également la possibilité d'adapter aux besoins de l'utilisateur les messages guide-opérateur et les messages d'aide et, si possible, de faire suivre ces derniers de la liste des réponses permises. Une procédure de vérification des erreurs qui utilise le dictionnaire est également fournie comme outil de mise au point.

Le présent mémoire donne une description détaillée de chacune des procédures, y comprise l'algorithme utilisé et une explication de son utilisation, suivie d'exemples. Le progiciel GETWRD est écrit en FORTRAN ANSI, norme de 1977. On s'est appliqué à le rendre aussi transférable qui possible.

GETWRD Package

Table of Contents

Section	Page
Abstract	i
1. INTRODUCTION	1
1.1 Implementation of a Command Language	1
1.2 Characteristics of a friendly program/user interface	1
2. FEATURES OF THE GETWRD PACKAGE	2
3. USE OF THE GETWRD PACKAGE	6
3.1 Accessing the GETWRD Package	6
3.2 Interfacing with the GETWRD Package	6
3.2.1 Using CLBUFF to initialize the input buffer	6
3.2.2 Using DICCHK to check the dictionary	7
3.2.3 Using GETWRD to return a word of user input	8
3.2.4 Using GETNUM to return a number	10
3.2.5 Using NXTWRD to check for more user-input	11
4. SAMPLE PROGRAMS AND OUTPUT	11
5. USER CUSTOMIZATIONS	18
5.1 Modifying the default Special Characters	18
5.2 Modifying the default input/output devices	19
6. CONCLUDING REMARKS	19
Appendix	Page
A. ALPHABETIC ORDER	20
B. THE GETWRD PACKAGE ENVIRONMENT	21
C. DATA STRUCTURES USED IN THE GETWRD PACKAGE	22
D. INDIVIDUAL PROCEDURES IN GETWRD PACKAGE	23
D.1 Subroutine CLBUFF	23
D.2 Subroutine DELETE	23
D.3 Logical Function DICCHK	24
D.4 Logical Function DLIMIT	26
D.5 Logical Function GETNUM	26
D.6 Logical Function GETWRD	28
D.7 Subroutine INSBUF	28

GETWRD Package

D.8 Logical Function MDICW	29
D.9 Logical Function NXTWRD	30
D.10 Logical Function PRMPT	30
D.11 Subroutine UPCASE	31
D.12 Logical Function WRDTRM	31
E. MACHINE DEPENDENCIES	33
Subject Index	34



1 INTRODUCTION

In any strongly interactive computer program, considerable effort is expended to provide an adequate interface between the user and the program. FORTRAN has been notorious for its poor capability for interaction with users. The GETWRD Package provides, within FORTRAN, a facility which interacts with the user to return a valid word of user input from a list of possible valid responses. It is written in FORTRAN 7.0 on a DEC-20 computer (see Appendix B) at the Defence Research Establishment Atlantic (DREA) for the purpose of implementing an effective interface for use with interactive programs.

1.1 Implementation of a Command Language

In any interactive program where the user has a variety of options from which to choose, prompting for each possible choice often becomes a tedious process. For example, consider the situation where a large calculation, which requires many user specified parameters, is to be performed repeatedly. Often between successive runs, only a few of these parameters need to be changed. Rather than prompting the user for each of the many parameters before each run, it would be quicker if the user could tell the program which parameters to change. Parameters not explicitly changed would remain as they were. A command language provides such a facility. A command language is useful wherever there is a large number of options because it gives the user the capability to tell the program what to do immediately rather than waiting passively for the proper prompt.

The GETWRD Package has been designed to ease the implementation of a command language in a large program. It is a collection of procedures which can be used to obtain user input. Use of the GETWRD Package saves the implementor of the large program from designing and coding new input routines.

1.2 Characteristics of a friendly program/user interface

A friendly program/user interface is one that is flexible and easy to use. One essential ingredient is that a user be given adequate prompting so that he is able to respond correctly. Secondly, his response should be tested to detect errors, and his mistakes should not be fatal. Usually a short prompt is sufficient to trigger a correct response from an expert user. However, new users appreciate a program where, if need be, they can request elaboration of a terse prompt. In the situation where the response expected from the user must be one of a number of possible responses, the computer program should be even better able to help the user. If the user can inquire about the possible responses, then he will be better able to respond correctly. Also, since there is a specific list of responses, type-ahead, word completion and spelling correction can be provided. These characteristics can make an interactive computer program much more tolerant of a user's mistakes or lack of familiarity with the program. The GETWRD Package has been designed with these characteristics in mind.

2 FEATURES OF THE GETWRD PACKAGE

The GETWRD Package is able to interpret a *word* of user input by matching it with one of the entries in a user-supplied dictionary. In this context, a *word* is defined to be any sequence of characters delimited by special *Delimiting characters*. The usual delimiting character is a blank, but up to five other delimiting characters may be defined (see Section 5.1). Alternatively, the GETWRD Package is able to interpret a *word* of user input as a real number. A user-supplied dictionary is not used in this case.

Whenever a *word* of user input is required in the program which is using the GETWRD Package, the program calls a GETWRD Package procedure to obtain the required input. A buffer is used as an intermediate storage area for all user input. Thus, although for each call to the GETWRD Package procedures, only one word of user input is examined, many words may be input by the user at a single time. They will be stored in the buffer and processed sequentially in subsequent calls to the GETWRD Package procedures. Only when there is no unprocessed data in the buffer, or an input error is encountered, will the program prompt the user for more input. This provides a type-ahead facility for an experienced user who is able to input his expected responses quickly without waiting for individual prompts. A user's input is stored in the buffer after all previously inserted input, unless the buffer has been re-initialized.

The principle GETWRD Package procedure is GETWRD. When called, GETWRD first checks the buffer to see if it contains any unprocessed words. If not, it prompts the user for more input. GETWRD then examines the first unprocessed word of user input and tries to match it with one of the words in a dictionary of possible valid responses. If it is successful, it returns to the calling program which then acts according to which word in the dictionary has been matched. If it is not successful, an error message is given, and the user is prompted for more input. Control is not passed back to the calling program until a word in the dictionary has been matched.

A word of user input is considered to match a dictionary word if the letters of the user word uniquely match the first letters of the dictionary word. That is, if the user inputs a word of 5 letters, these must be the first 5 letters of the dictionary word which it matches, and no other word in the dictionary can start with those 5 letters. For example, if the words in the dictionary are BEAR, BEETLE, ELEPHANT, and TIGER, then the user input 'T' is sufficient to match with the dictionary word TIGER. However, 'BE' does not match any dictionary word since both BEAR and BEETLE begin with 'BE'. However, 'BEA' will match BEAR and 'BEE' will match BEETLE. Similarly, 'BEER' will not match any of the dictionary words.

Numbers may also be interpreted as commands, but this is done via the GETWRD Package procedure called GETNUM. When GETNUM is called, it too examines the buffer for the next unprocessed word. If it cannot find one, it prompts the user for more input. GETNUM then interprets the word as a number. No dictionary is necessary for GETNUM. If the word cannot be interpreted as a number (for example, if it is a list of alphabetic characters rather than numerals) then an error message is given and the user is prompted for more input. Note that since both GETWRD and GETNUM use the same buffer, the type-ahead facility is not affected if some words of input are character strings and some are numbers.

In order to increase the friendliness of the GETWRD Package, several characters have been defined to have special meanings. If the user needs help in responding to a

prompt, he may type a *Help character* (Default is "?"). This causes a helpful message followed by a list of all possible words in the dictionary to be printed. Should the user request help (by typing "?") in the middle of his word, only those words in the dictionary which match his current word will be displayed. For example, with the dictionary of animals above, the input '?' would cause the following message to be displayed:

```
This is a help message created by the programmer.  
BEAR  
BEETLE  
ELEPHANT  
TIGER
```

However, the input 'BE?' would only cause the following to be displayed:

```
This is a help message created by the programmer.  
BEAR  
BEETLE
```

since only BEAR and BEETLE match the current word 'BE'.

Another special character is the *Completion character* (Default is "\$" or <dollar sign>). If the current word typed by the user matches a unique word in the dictionary, the GETWRD Package will complete his word in the buffer with the dictionary word. For example, if the input is 'T\$', the contents of the buffer will be changed to 'TIGER' since only TIGER matches the current word 'T'. This feature is useful when subcommands are used. For example, consider the following 3-tiered command set:

<u>1st command</u>	<u>2nd command</u>	<u>3rd command</u>
WASH	BEAR	NOW
FEED	BEETLE	LATER
	ELEPHANT	TOMORROW
	TIGER	

Suppose the user types in two commands:

W E<CR>

(<CR> denotes a carriage return). This is sufficient for GETWRD to match with the commands WASH ELEPHANT. However, a third level command is still expected by the program. Therefore, GETWRD prompts the user for more input by typing a prompt symbol (here the prompt symbol is '=>') followed by the full contents of the buffer.

=>W E

In response, the user now inputs a third level command. However, suppose that the *Completion character* had been included with the simple one letter commands above. That is, suppose the user had typed

W\$ E\$<CR>

The commands "WASH ELEPHANT" would still have been matched. However, the prompt for a third level command would have been

=>WASH ELEPHANT

The difference in the prompting occurs because the *Completion character* causes the commands WASH and ELEPHANT to be completed in the buffer. Thus, the *Completion character* is a simple means of obtaining more intelligible prompts. An experienced user will rarely want this feature, but it is very useful for the novice.

The *Completion character* is also very useful if two dictionary words have identical letters at their beginnings. For example, suppose the dictionary contains the words ELEPHANT-LOVER and ELEPHANT-HATER. A user making frequent use of these words will not wish to type in 'ELEPHANT-' many times. With the *Completion character* he may type 'E\$'. Presuming that the dictionary contains no other words beginning with 'E', GETWRD will respond

?? Word completion not unique

=>ELEPHANT-

The possible valid words have been completed until the first discrepancy in the match. The user need only type 'L' or 'H' to distinguish between the two commands.

Two other special characters are defined to allow the user to erase his typing mistakes. They are the *Delete Letter character* and the *Delete Word character*. The former causes the previous letter in the buffer to be deleted. The latter causes the current word typed by the user to be deleted; that is, letters are deleted backwards until a delimiter is reached. The default for *Delete Letter character* is a "\ " and the default for *Delete Word character* is a "<".

One final special character is the *Abort character* (Default is "^" or <hat sign> character). Typing it will cause control to return to the user's calling program. Typing a <CNTRL>C (on a DEC-20 computer) will cause control to be returned to the monitor.

The GETWRD Package is also easy to use for the programmer. Figure 1 is a schematic diagram of a very simple program illustrating the use of the GETWRD Package. The program governs the movements of two simple geometric objects, a star and a box. At the highest level of the command structure there are three commands:

- EXIT - Causes the program to terminate
- MOVE - Causes one of the objects to be moved
- PLOT - Causes the positions of the objects to be plotted

The command MOVE has several subcommands. The first level of subcommand indicates the object to be moved. The subcommands are BOX and STAR. The second level of subcommand indicates the direction in which the object is to be moved. The possible sub-sub-commands are DOWN, LEFT, RIGHT, and UP. The final level of subcommand indicates the distance the object must be moved. Since this is a number, there is no dictionary.

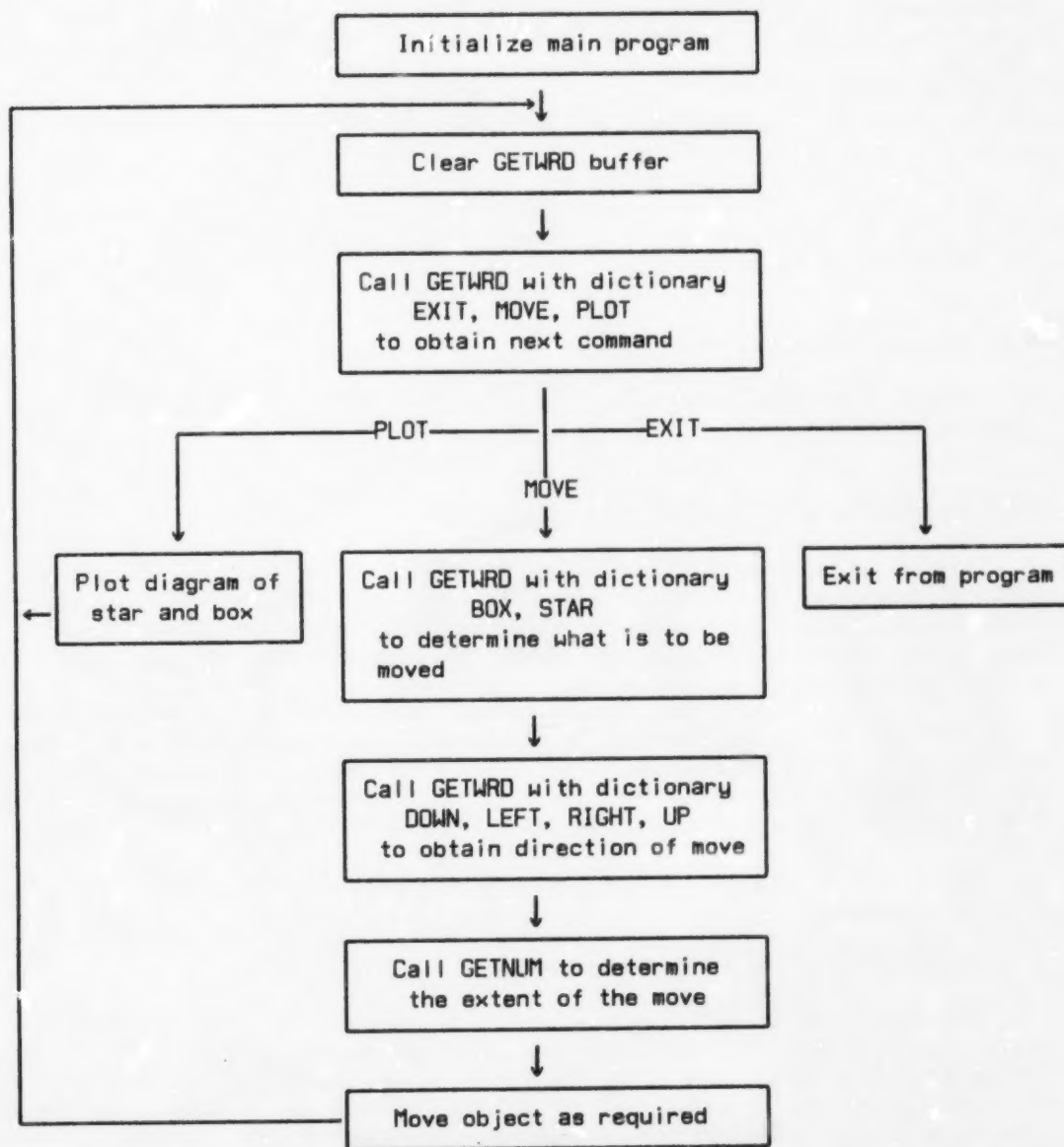


Figure 1. Schematic diagram of a sample program using the GETWRD Package

Every time the programmer requires an additional word of input from the user he calls either GETWRD or GETNUM, depending on whether the input is a word or a number. Notice that the dictionary may be different each time GETWRD is called. Also notice that when control returns to the highest level of command, the buffer is cleared to prevent overflow.

This example, in a slightly expanded version, will be examined in detail in Section 4.

3 USE OF THE GETWRD PACKAGE

3.1 Accessing the GETWRD Package

The GETWRD Package resides in the file PS:<HALLY>GETWRD.REL on the DEC-20 computer at DREA.

3.2 Interfacing with the GETWRD Package

Any of the procedures defined within the GETWRD Package are used via the usual FORTRAN calling sequence. The user must remember to define and initialize, within the calling program, any of the variables used as parameters for a procedure. In particular, when using a function rather than a subroutine, the function name itself must be declared to be either integer, real or logical.

The GETWRD Package contains twelve procedures in all, but there are only five which are likely to be called explicitly by the main program : the subroutine CLBUFF, and the functions DICCHK, GETNUM, GETWRD and NXTWRD. The others are called by these five, but will not find general application outside their use within the GETWRD Package.

The use of the five special procedures is now examined in more detail.

3.2.1 Using CLBUFF to initialize the input buffer

The subroutine CLBUFF (see Appendix D.1) clears all or part of the buffer by setting its contents to the *Padding character*. The *Padding character* is the character which FORTRAN uses to pad a character string when only part of the string is assigned. FORTRAN 77 standard requires that the *Padding character* be a space. The *Padding character* is always considered to be a delimiting character in addition to those set by the user (see Section 5.1). Whenever the contents of all or part of the buffer are to be discarded, CLBUFF must be invoked. In particular, CLBUFF is used to initialize the buffer so that subsequent input will not cause an overflow (see Figure 1).

Associated with the buffer is a pointer to the position currently being examined within the buffer. This pointer should be set to zero whenever the buffer is cleared so that subsequent input is put at the beginning of the buffer.

3.2.2 Using DICCHK to check the dictionary

The logical function DICCHK checks the dictionary for errors, returning TRUE if no errors are found and FALSE otherwise. DICCHK is intended as a debugging tool and, if the dictionary is not being changed during the execution of the program, may be removed from the program for the sake of efficiency once the dictionary is known to be error free.

The arguments of DICCHK are NWRDS, the number of dictionary words, and DICT, the array of character variables containing the dictionary words. DICCHK checks for the following errors in the dictionary (see Appendix C for a description of the required dictionary format):

- 1) $NWRDS < 0$. The variable NWRDS must be a positive integer.
- 2) One of the dictionary words consists only of the *Padding character* (see Section 3.2.1 and Appendix C).
- 3) One of the dictionary words contains *Padding characters* at the beginning or in the middle of the word. *Padding characters* are allowed only at the end of the dictionary words.
- 4) One of the dictionary words contains a delimiter which is not a *Padding character*.
- 5) One of the dictionary word contains lower case letters.
- 6) One of the dictionary words contains one of the special characters: the *Abort character*, the *Completion character*, the *Delete Letter character*, the *Delete Word character*, or the *Help character*.
- 7) The dictionary is not in alphabetic order (see Appendix A).
- 8) Two words in the dictionary are identical.

The following are the essential elements of the use of DICCHK in a program:

```
PROGRAM MAIN
```

```
  .  
  .  
  .
```

C Remember to declare variables and the function name, GETWRD.

```
CHARACTER DICT(3)*4  
INTEGER NWRDS  
LOGICAL DICCHK
```

C Set up the dictionary

```
DATA DICT/'EXIT','MOVE','PLOT'/  
DATA NWRDS /3/
```

C Check the dictionary. If there are errors, stop. Otherwise continue.

```
IF (.NOT.DICCHK(NWRDS,DICT)) STOP
.
.
END
```

3.2.3 Using GETWRD to return a word of user input

The function GETWRD (see Appendix D.6) interacts with a user to *get the next word*¹ by trying to match the user's response to one of the entries in the user-supplied dictionary. The dictionary is a list of *words* in alphabetic order (see Appendix A) and in upper case letters. The matching process is letter case independent because GETWRD converts all user input to upper case before comparing it with the list of valid words. If the user's response is valid, GETWRD returns TRUE. The value of DICPOS then indicates the position in the dictionary of the word identified. Thus, if the dictionary is

```
BEAR
BEETLE
ELEPHANT
TIGER
```

and the user input is 'eleph', then GETWRD returns TRUE and DICPOS = 3, indicating that 'eleph' matches the third word in the dictionary, 'ELEPHANT'. Notice that lower case input will still match the dictionary words.

Normally, GETWRD continues to prompt the user until a match is found. However, there are two cases in which it will return with no match. In both cases, GETWRD returns FALSE. The value of DICPOS then indicates the type of error involved. Returning DICPOS = -1 acts as a controlled abort signal (see Section 2). Returning DICPOS = -2 indicates that the buffer has overflowed.

GETWRD is a logical function so it must be explicitly declared to be such within the calling program. GETWRD also requires the use of a buffer, a dictionary of *words*, a prompt, and a help message (see Appendices C and D.6).

As an example, assume that in response to the prompt "=>", the user is required to input one of the following commands:

- 1) EXIT - exit from program
- 2) MOVE - perform a MOVE action
- 3) PLOT - perform a PLOT action

A useful help message might be 'Enter one of the following commands'. The following are the essential elements of a single call to the integer function, GETWRD:

¹ For getting a number rather than a word, see Section 3.2.4.

PROGRAM MAIN

.
.
.

C Remember to declare variables & the function name, GETWRD.

CHARACTER BUFFER*80, DICT(3)*4, HLPMSG*26, PROMPT*2
INTEGER BUFPOS, NBUFF, NWRDS
LOGICAL GETWRD

C Set up the dictionary

DATA DICT/'EXIT','MOVE','PLOT'/

C Set up help message

DATA HLPMSG /'ENTER ONE OF THE FOLLOWING COMMANDS'/

C Set up prompt in A1 format

DATA PROMPT/'->'/
DATA NWRDS /3/

CALL CLBUFF (BUFFER,1,NBUFF)

! Initialize BUFFER

BUFPOS=0

! Initialize BUFFER pointer

C Get next word from user

IF (GETWRD (BUFFER,NWRDS,DICT,PROMPT,HLPMSG,BUFPOS)) THEN

C The user input has been matched with DICT(DICPOS)

.
.
.

ELSE

C There is a controlled abort or buffer overflow

.
.
.

END IF

.
.
.

END

3.2.4 Using GETNUM to return a number

The function GETNUM (see Appendix D.5) interacts with a user to return a real number. It understands the input of an integer by the user but always converts it into a real number. GETNUM reads the next word in the buffer, assumes that it is a numerical value, and converts it into G15.7 format from A format. If GETNUM succeeds in interpreting the input as a real number, GETNUM returns TRUE. The value of the number is contained in the variable RNUM. Normally, GETNUM will continue to reprompt the user until a valid number is obtained. However, there are two cases in which GETNUM will return before a valid number is obtained. In both cases, GETNUM returns FALSE. The value of the argument IER then indicates the type of error involved. Returning IER = -1 acts as a controlled abort signal (see Section 2). Returning IER = -2 indicates that the buffer has overflowed.

GETNUM is a logical function so it must be explicitly declared to be such within the calling program. Unlike GETWRD, GETNUM does not use a dictionary so word completion and spelling correction are not provided. However, like GETWRD, GETNUM does require the use of a buffer, a prompt, and a help message (see Appendix C).

As an example, assume that in response to the prompt 'enter your age: ', the user is required to input his age. A useful help message might be 'Enter an integer value between 1 and 99 (if older than 99, then enter 99)'. Following are the essential elements of a single call to the real function, GETNUM:

```
PROGRAM MAIN
```

```
  .  
  .  
  .
```

C Remember to declare variables & the function name, GETNUM

```
CHARACTER BUFFER*10,HLPMSG*74,PROMPT*16  
INTEGER BUFPOS,IER  
LOGICAL GETNUM  
REAL AGE
```

C Notice! No dictionary is used in GETNUM !

C Set up prompt

```
DATA PROMPT/'ENTER YOUR AGE: '/
```

C Set up help message

```
HLPMSG = 'ENTER AN INTEGER VALUE BETWEEN 1 AND 99 '  
*  '/' (IF OLDER THAN 99, THEN ENTER 99).'
```

```
CALL CLBUFF (BUFFER,1,NBUFF)  
BUFPOS=0
```

```
! initialize BUFFER  
! initialize BUFFER pointer
```

```
  .  
  .  
  .
```

C Prompt for age and store response in AGE

```
IF (GETNUM (BUFFER, PROMPT, HLPMSG, BUFPOS, AGE, IER)) THEN
```

C A value for AGE has been found

```
.
.
.
```

C There is a controlled abort or a buffer overflow

```
ELSE
```

```
.
.
```

```
END IF
```

```
.
.
```

```
END
```

3.2.5 Using NXTWRD to check for more user-input

As a result of the type-ahead facility, the buffer may contain more user input than was required in response to the previous prompt. The logical function NXTWRD (Appendix D.9) checks for such unprocessed input but it does not prompt for more input. NXTWRD returns TRUE only if there is still an unprocessed word (or number) in the buffer. This feature is useful in the implementation of special options for a command. For example, assume that

"=>DIRECTORY" means "LIST the files on my directory"

and that

"=>DIRECTORY ARCHIVE" means "LIST the ARCHIVED files on my directory".

The NXTWRD function would be used to check for the option ARCHIVE before performing the directory listing, but it would not prompt the user for more input as would the GETWRD function. Section 4 contains a more complete example of the use of NXTWRD.

4 SAMPLE PROGRAMS AND OUTPUT

Two sample programs illustrating the main features of the GETWRD Package are presented in this section. The first, called DICMN, illustrates the use of the dictionary checking procedure DICCHK.

PROGRAM DICMN

```
C-----C
C Example program displaying the features of DICCHK C
C-----C
```

CHARACTER DICT(8)*5, DLMTR*5

```

      INTEGER NWRDS
      LOGICAL DICCHK
      COMMON / DLMSTR / DLMTR
      DATA DICT/' ','(AND)','move?','MOV','EXIT','EXIT',' PLOT',
*      'TH EN'/
      DATA NWRDS /8/

      DLMTR = ' () []'
      IF (DICCHK(NWRDS,DICT)) THEN
        WRITE(5,'(/A)') ' Dictionary OK...'
      ELSE
        WRITE(5,'(/A)') ' Dictionary contains errors...'
      END IF
      STOP
      END

```

The following is the output from a run of the program DICMN on the DEC-20 computer at DREA.

```

●EXE DICMN,PS:<HALLY>GETWRD
LINK: Loading
[LNKXCT DICMN Execution]

```

```

??DICTIONARY ERROR : Null word
DICT(1) = ' '

```

```

??DICTIONARY ERROR : Word contains delimiter other than PADCHR
DICT(2) = '(AND)'
      ^

```

```

??DICTIONARY ERROR : Word contains delimiter other than PADCHR
DICT(2) = '(AND)'
      ^

```

```

??DICTIONARY ERROR : Lower case letters in word
DICT(3) = 'move?'
      ^

```

```

??DICTIONARY ERROR : Word contains one of the special characters
DICT(3) = 'move?'
      ^

```

```

??DICTIONARY ERROR : Dictionary not in alphabetical order
DICT(3) = 'move?'
DICT(4) = 'MOV '

```

```

??DICTIONARY ERROR : Dictionary not in alphabetical order
DICT(4) = 'MOV '
DICT(5) = 'EXIT '

```

```

??DICTIONARY ERROR : Two dictionary words are identical
DICT(5) = 'EXIT '
DICT(6) = 'EXIT '

```

??DICTIONARY ERROR : PADCHR's are allowed only at the ends of words
 DICT(7) = ' PLOT'
 ^

??DICTIONARY ERROR : Dictionary not in alphabetical order
 DICT(6) = 'EXIT '
 DICT(7) = ' PLOT'

??DICTIONARY ERROR : PADCHR's are allowed only at the ends of words
 DICT(8) = 'TH EN'
 ^

Dictionary contains errors...
 CPU time 0.48 Elapsed time 0.77

The second sample program, called PLACE, illustrates the implementation of a command language using the GETWRD Package. It allows the user to manipulate two simple objects, a box and an asterisk, in a 10x10 field².

PROGRAM PLACE

```

C-----C
C Example program displaying the features of the GETWRD Package C
C C
C Allows the manipulation of simple objects drawn in a 10X10 C
C field. C
C-----C
CHARACTER BUFFER*80,DICT(5)*4,HLPMSG*26,PROMPT*2
INTEGER BUFPOS,NWRDS,DICPOS
LOGICAL NXTWRD,GETWRD

DATA DICT/'AND','EXIT','MOVE','PLOT','THEN'/, PROMPT /'=>'/,
* HLPMSG /'ENTER ONE OF THE FOLLOWING COMMANDS'/, NWRDS /5/

5 CALL CLBUFF (BUFFER,1,80)
  BUFPOS=0

10 IF (.NOT.GETWRD (BUFFER,NWRDS,DICT,PROMPT,HLPMSG,BUFPOS,DICPOS))
  * GO TO 5

  IF (DICT(DICPOS).EQ.'EXIT') THEN
    WRITE (5,'(A)') 'Exiting ...'
    STOP
  ELSE IF (DICT(DICPOS).EQ.'MOVE') THEN
    CALL MOVE (BUFFER,BUFPOS)
  ELSE IF (DICT(DICPOS).EQ.'PLOT') THEN
    CALL PLOT
  END IF

  IF (NXTWRD (BUFFER,BUFPOS+1)) GO TO 10
  GO TO 5

```

² The top-level commands AND and THEN and the word THE in the dictionary for the procedure MOVE, serve no function other than to allow more readable commands. They may be disregarded by the user if desired.

STOP
END

SUBROUTINE MOVE (BUFFER,BUFPOS)

```

C-----C
C An object is to be moved. First GETWRD is called to determine C
C whether the square or the star is to be moved. Then C
C subroutine WHERE is called to determine where it is to be C
C moved. C
C-----C
      DIMENSION BUFFER*80,DICT(5)*8,HLPMSG*34,PROMPT*7
      INTEGER BUFPOS,NWRDS,DICPOS
      LOGICAL GETWRD

      COMMON / SQUARE / ISQRX,ISQRY
      COMMON / STAR / ISTRX,ISTRV

      DATA DICT/'ASTERISK','BOX','SQUARE','STAR','THE'/,
* HLPMSG /'ENTER ONE OF THE FOLLOWING SYMBOLS'/,
* PROMPT/'shape=>'/, NWRDS /5/

      IF (.NOT.GETWRD (BUFFER,NWRDS,DICT,PROMPT,HLPMSG,BUFPOS,DICPOS))
* RETURN

      IF ((DICT(DICPOS).EQ.'BOX').OR.(DICT(DICPOS).EQ.'SQUARE')) THEN
          CALL WHERE(ISQRX,ISQRY)
      ELSE IF ((DICT(DICPOS).EQ.'ASTERISK').OR.
* (DICT(DICPOS).EQ.'STAR')) THEN
          CALL WHERE(ISTRX,ISTRV)
      END IF
      RETURN

      END
  
```

SUBROUTINE WHERE (BUFFER,BUFPOS,IX,IY)

```

C-----C
C GETWRD is called to determine in which direction the object C
C is to be moved. GETNUM is then called to determine the number C
C of spaces the object is to be moved. The object is then moved C
C the appropriate number of spaces. C
C-----C
      CHARACTER BUFFER*80,DICT(4)*5,HLPMSG*37,HLP1*33,
* PROMPT*7,PMPT1*8
      INTEGER BUFPOS,IMOV,NWRDS,DICPOS,IER
      LOGICAL GETWRD,GETNUM
      COMMON / MSGS / PROMPT,HLPMSG
      DATA DICT/'DOWN','LEFT','RIGHT','UP'/,PROMPT/'where=>'/,
* HLPMSG /'ENTER ONE OF THE FOLLOWING DIRECTIONS'/,
* HLP1 /'ENTER AN INTEGER BETWEEN 1 AND 10'/,PMPT1/'number=>'/,
* NWRDS /4/
      DATA ISQRX /5/,ISQRY /5/,ISTRX/1/,ISTRV/1/

      IF (.NOT.GETWRD (BUFFER,NWRDS,DICT,PROMPT,HLPMSG,BUFPOS,DICPOS))
* RETURN
  
```

```

      IF (.NOT.GETNUM (BUFFER,PMPT1,HLP1,BUFPOS,RMOV,IER)) RETURN
C Process the user's response
      IF (DICT(DICPOS).EQ.'DOWN') THEN
        .
        .
      END
      SUBROUTINE PLOT
C-----C
C The objects are plotted in a 10X10 field      C
C-----C
      DIMENSION FIELD(10,10)
      COMMON / SQUARE / ISQRX,ISQRY
      COMMON / STAR / ISTRX,ISTR
        .
        .
      END

```

Following is output from a typical run of the program PLACE. Input from the user appears as lower case letters to the right of the prompt sign, =>, whereas upper case letters to the right of the prompt is computer output. The symbol '\$' represents an actual dollar sign, NOT the <escape> or <alt> character.

```

@EXE PLACE,PS:<HALLY>GETWRD
LINK: Loading
[LNKXCT PLACE Execution ]

```

```

=>?

```

```

ENTER ONE OF THE FOLLOWING COMMANDS
AND
EXIT
MOVE
PLOT
THEN

```

```

=>plot

```


GETWRD Package

shape=>M S?

SQUARE

STAR

shape=>M Sq d 3 then plot

*

XXX
XXX

```
=>move asterisk up<down 4 then move the square right 2 then plot
```

*
 XXX
 XXX

=>move the st\q*

where $\theta \Rightarrow$ MOVE THE SQUARE to 3

?? Word not in dictionary

where=>MOVE THE SQUARE Up 3 then plot and exit


```
COMMON / ENDCHR / HLPCHR,CMPLT,DELLTR,DELWRD,ABORT
```

```
COMMON / DLMSTR / DLMTR*(NDLMTR)
```

where

HLPCHR is the *Help* character,

CMPLT is the *Completion* character,

DELLTR is the *Delete Letter* character,

DELWRD is the *Delete Word* character,

ABORT is the controlled *Abort* character,

DLMTR is a character string of length NDLMTR containing the possible *Delimiting* characters. Default has all delimiting characters equal to blanks.

NDLMTR is a parameter equal to the number of allowed delimiting characters in addition to the *Padding* character (see Section 3.2.1) which is always a delimiter. This number is set in a parameter statement in the procedure DLIMIT. Its current value is 5.

It is important to note that the default characters "?", "\", "<", and "^" are not in the ANSI standard character set and may not be allowed on some computers.

5.2 Modifying the default input/output devices

The input and output devices are both defined to be UNIT 5 in a DATA statement within the GETWRD Package. However, they may be changed by passing different device numbers via the COMMON block called IOUNIT as follows:

```
COMMON / IOUNIT / UNITIN,UNTOUT
```

where UNITIN and UNTOUT are integer variables.

6 CONCLUDING REMARKS

The GETWRD Package provides, within FORTRAN, a friendly, easily programmed interface between the user and the program. It allows the user to obtain help messages and lists of allowed responses, to use abbreviated input, and to complete input using word completion. These features make it especially useful during the implementation of a command language for a large program.

Care has been taken to make the package as portable as possible so that a minimum number of changes need be made when using it at a site other than DREA.

Appendix A

ALPHABETIC ORDER

The following is a list of the ASCII characters which are allowed as letters in a dictionary word. They are listed (from left to right) in alphabetical order. It is important to note that, although this sequence is not the collating sequence used on some computers, it is the ANSI FORTRAN 77 standard sequence for use with the character string comparison functions LGE, LGT, LLE and LLT, which are used in the GETWRD Package for comparing user input with the dictionary words.

space	!	"	#	\$	%	&	'
(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7
8	9	:	;	<	=	>	?
•	A	B	C	D	E	F	G
H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W
X	Y	Z	[	\	]	^	_
{		}	~				

Appendix B**THE GETWRD PACKAGE ENVIRONMENT**

The GETWRD Package is written in FORTRAN 7.0 on the DEC-20 computer at the Defence Research Establishment Atlantic (DREA). To the authors knowledge, it conforms to the ANSI Standard FORTRAN of 1977, except where noted explicitly (see Appendix E). At DREA, a copy of the source file may be found in PS:<HALLY>GETWRD.FOR. Users at other sites may obtain a copy of the source file from the authors.

Appendix C

DATA STRUCTURES USED IN THE GETWRD PACKAGE

The following are descriptions of the data structures used throughout the GETWRD Package. Additional variables which are unique to a particular procedure will be explained in Appendix D.

- 1) The buffer is defined to be a character string. Within the GETWRD Package, the buffer is referred to by the variable, BUFFER. Its length is determined by a declaration statement in the calling program and is determined within the GETWRD Package by the implicit FORTRAN function LEN. Where the length of BUFFER is stored explicitly in a variable (procedures CLBUFF, INSBUFF, and PRMPT), the variable used is NBUFF. A pointer to the current position in the buffer is stored in the integer variable BUFPOS. The position in the buffer of the first character in the word currently being processed is stored in the integer variable BEGWRD.
- 2) The dictionary is stored in an array of character strings. Each element in the array contains one possible word to be identified as the next word of user input. ALL dictionary words must be in upper case because user input is converted to upper case before being compared to the words in the dictionary. The matching function, MDICW, requires that the words in the dictionary be sorted in alphabetical order (see Appendix A for the correct order of non-alphabetic characters). The only *Delimiting character* allowed in a dictionary word is the *Padding character*, the character used by FORTRAN to pad character arrays which are only partly assigned. FORTRAN 77 standard requires this character to be a blank. Within the GETWRD Package, the dictionary is referred to by the variable, DICT, and its array length is set by the variable NWRDS; that is, NWRDS is the number of words in the dictionary. The string length of each element of DICT is set by a declaration statement in the calling program and is determined within the GETWRD Package by the implicit FORTRAN function LEN. A pointer to the word in the dictionary currently being examined is stored in the integer variable DICPOS.
- 3) The *message* to be used to prompt the user for input is a character string. Within the GETWRD Package, this prompt is stored in the character variable PROMPT. Its length is set by a declaration statement in the calling program and is determined within the GETWRD Package by the implicit FORTRAN function LEN.
- 4) The *explanation* to be used to help the user when he requests help is a character string. Within the GETWRD Package, this help message is stored in the character variable HLPMSG. Its length is set by a declaration statement in the calling program and is determined within the GETWRD Package by the implicit FORTRAN function LEN.

Appendix D

INDIVIDUAL PROCEDURES IN GETWRD PACKAGE

This section documents each of the individual procedures defined within the GETWRD Package. The purpose, argument list, local variables and an outline of the algorithm is given for each individual procedure.

D.1 Subroutine CLBUFF

Purpose

CLBUFF deletes the characters in the buffer between positions LOBUFP and HIBUFP inclusive, and left justifies characters (HIBUFP+1 to NBUFF) to LOBUFP. The end of the buffer is then padded with the *Padding character*.

Arguments: BUFFER, LOBUFP, HIBUFP

where

BUFFER (see Appendix C)

LOBUFP = The position in the buffer of the first character to be cleared

HIBUFP = The position in the buffer of the last character to be cleared

Common block: COMMON / DLMSTR / DLMTR (see Section 5.1)

Algorithm:

```
Begin
  NBUFF = Length of BUFFER
  If HIBUFP < NBUFF then
    Left justify characters (HIBUFP+1 to NBUFF) to LOBUFP
  End if
  Set each character from NBUFF-HIBUFP+LOBUFP to NBUFF
  equal to the Padding character
End
```

D.2 Subroutine DELETE

Purpose

DELETE processes any delete characters in the input buffer from the current buffer position (BUFPOS+1) to the end of the buffer. If the delete character is the *Delete Letter character*, then that delete character and the previous character are deleted. If the delete character is *Delete Word character*, then the current word is deleted.

Arguments: BUFFER, BUFPOS, BEGWRD (see Appendix C)

Common block: COMMON / ENDCHR / HLPCHR, CMPLT, DELLTR, DELWRD, ABORT
(see Section 5.1)

Internal variables:

DELPOS = Position in BUFFER of the next *Delete Letter character* or
Delete Word character

Algorithm:

Begin

While there is another *Delete Letter character* beyond BEGWRD in
BUFFER do

DELPOS = position in BUFFER of *Delete Letter character*

If DELPOS = BEGWRD then

Delete the *Delete Letter character* only

Else

Delete the *Delete Letter character* and the previous character

If BUFPOS points to the character immediately preceding
the *Delete Letter character* then BUFPOS = BUFPOS-1

End if

End do

While there is another *Delete Word character* beyond BUFPOS in
BUFFER do

DELPOS = position in BUFFER of *Delete Word character*

Find the first *Delimiting character* at or preceding DELPOS

Delete all characters from DELPOS back to but not including the
Delimiting character

If the *Delimiting character* precedes BUFPOS then set BUFPOS
to the position in BUFFER of the *Delimiting character*

End do

End

D.3 Logical Function DICCHK

Purpose

DICCHK is used to check the dictionary for errors. It is intended as a debugging
tool.

Arguments: NWRDS, DICT (see Appendix C)

Common blocks:

COMMON / ENDCHR / ENDC (see Section 5.1)

COMMON / IOUNIT / UNITNO (see Section 5.2)

where

ENDC = An array length 5 each of whose elements is a character string of length 1, containing the *Abort character*, the *Completion character*, the *Delete Letter character*, the *Delete Word character*, and the *Help character*.

Algorithm:

```
Begin
  DICCHK=.TRUE.
  If NWRDS < 0 then
    Write error message
    DICCHK=FALSE
  Else
    While not at end of dictionary do begin
      Get next dictionary word
      If word is null then
        Write error message
        DICCHK=FALSE
      Else
        While not at end of word do begin
          Get next letter in word
          If letter is Padding character then
            If remaining letters are not Padding characters then
              Write error message
              DICCHK=FALSE
            End if
          Else if letter in a delimiter then
            Write error message
            DICCHK=FALSE
          Else if letter is lower case then
            Write error message
            DICCHK=FALSE
          Else if letter is one of the characters in ENDC,
            i.e. one of the special characters, then
            Write error message
            DICCHK=FALSE
          End if
        End
      End if

      If word is not the first in the dictionary then
        If word precedes the previous word in alphabetic order then
          Write error message
          DICCHK=FALSE
        Else if word is the same as the previous word
          Write error message
          DICCHK=FALSE
        End if
      End if
    End
  End if
  Return DICCHK
End
```

D.4 Logical Function DLIMIT

Purpose

DLIMIT defines word delimiting characters. Six such characters may be defined. Five are the characters in the string DLIMIT in the COMMON block DLMSTR. By default these characters are all blanks, but may be changed by the user. The *Padding character* (see Section 3.2.1) is also interpreted as a delimiter. DLIMIT returns TRUE if CHAR is one of the delimiting characters. Otherwise it returns FALSE. The number of allowed delimiting characters is set by the value of NDLMTR which is defined in a parameter statement (see Section 5.1).

Argument: CHAR1

where

CHAR1 = a character string of length 1

Common block: COMMON / DLMSTR / DLMTR (see Section 5.1)

Algorithm:

```
Begin
  If CHAR1 is NOT one of the delimiting characters then
    DLIMIT = FALSE
  Else
    DLIMIT = TRUE
  Return DLIMIT
End
```

D.5 Logical Function GETNUM

Purpose

GETNUM interprets the next *word* of user input as a real number, returning TRUE if successful and FALSE if the user has signalled an abort (by entering the *Abort character*) or if the buffer has overflowed. Initially, the current buffer position, BUFPOS, is set at the delimiting character preceding the first digit. GETNUM reads the next word in BUFFER, checks for *Abort characters* and *Help characters* and, if it finds none, assumes that it is a numerical value, and reads it (format specifier G30.15) from BUFFER into the single precision real variable RNUM, which is returned to the calling program. A *Help character* will cause a help message to be written, while an *Abort character* will cause control to be returned to the calling program with the variable IER set to -1.

Arguments: BUFFER, PROMPT, HLPMSG, BUFPOS, RNUM, IER

where

BUFFER, PROMPT, HLPMSG, BUFPOS, RNUM, IER (see Appendix C)

RNUM = The value of the number input

IER = An error flag
 = -1 If there is a controlled abort
 = -2 If the buffer has overflowed

Common blocks:

COMMON / ENDCHR / HLPCHR, CMPLT, DELLTR, DELWRD, ABORT (see Section 5.1)
 COMMON / IOUNIT / UNITNO (see Section 5.2)

Internal Variables:

BEGWRD, NWRDS (see Appendix C)

NDPT = The number of decimal points in the number. If NDPT = 0, then the number must be converted as an integer. Otherwise, the number is converted as a real number.

Algorithm:

```

Begin
  Initialize BEGWRD
  Repeat
    NDPT=0
    While (no characters past the present point in BUFFER) do begin
      Prompt user
      Read input
    End
    Repeat
      Get next character
      If character is the Abort character then
        IER = -1
        RETURN
      Else if character is decimal point
        Then NDPT = NDPT + 1
    Until character is word terminator

    If character is a delimiter then
      If NDPT=0 then
        Convert number as an integer
      Else
        Convert number as a real number
      End if
      If an error occurs in conversion then
        Write error message
        Delete back to beginning of word
      End if
    Else process termination character
  Until valid number obtained
  Return
End

```

D.6 Logical Function GETWRD

Purpose

GETWRD returns the value TRUE if the user input has been matched with a unique *word* in the dictionary, and returns FALSE if an abort has been signalled or if the buffer has overflowed. The position in the dictionary of the word identified is contained in the integer variable DICPOS. If an abort has been signalled (that is, the user has entered the *Abort character*), then DICPOS will have the value -1. If the buffer has overflowed, DICPOS will have the value -2.

Arguments: BUFFER, NWRDS, DICT, PROMPT, HLPMSG, BUFPOS, DICPOS
(see Appendix C)

Internal Variables: BEGWRD (see Appendix C)

Algorithm:

```
Begin
  Initialize DICPOS to 1
  While no characters past present point in BUFFER do begin
    Prompt user
    Read input
  End

  Repeat
    Get next character
    If not end of word then
      Find first possible word in dictionary
      If no valid word can be found then
        Clear BUFFER from present position to end
        Prompt user
        Read input
      End If
    Else process word terminator
    End If
  Until finished word
  GETWRD = TRUE
  Return position of word in dictionary
End
```

D.7 Subroutine INSBUF

Purpose

INSBUF inserts the characters contained in the array WORD into the buffer. It is used during word completion.

Arguments: BUFFER, NCHR, WORD(NCHR), BUFPOS

where

BUFFER (see Appendix C)

BUFPOS = Position in BUFFER of first character to be inserted

NCHR = Number of characters in WORD

WORD = A character string containing the word to be inserted

Algorithm:

Begin

Make room in BUFFER so that following characters will not be overwritten

If enough room in BUFFER then insert the completion of WORD into BUFFER

End

D.8 Logical Function MDICW

Purpose

MDICW finds the first possible word in the dictionary which matches the given word so far and sets DICPOS to the position of that word in the dictionary. Dictionary must be sorted in alphabetical order for the sake of efficiency. If no matching word is found, then MDICW is returned as FALSE, else it is returned as TRUE.

Arguments: BUFFER, NWRDS, DICT, DICPOS, BEGWRD, BUFPOS
(see Appendix C)

Common block:

COMMON / ENDCHR / HLPCHR,CMPLT,DELLTR,DELWRD,ABORT (see Section 5.1)

Internal variables:

DICCHR = Pointer to characters within the current word

IDICP = Do loop Index. Indicates a position in the dictionary

LSTCHR = The number of characters between BEGWRD and BUFPOS

Algorithm:

Begin

Initialize LSTCHR

If DICPOS > NWRDS or if LSTCHR exceeds the length of the dictionary words then

Return MDICW = FALSE

Else if the current word is null then

Return MDICW = TRUE

Else

Repeat

If the BUFFER word matches the current dictionary word then

RETURN MDICW = TRUE

```
      Else if the BUFFER word does not exceed the current dictionary
        word then
          RETURN MDICW = FALSE
        End if
      Get the next dictionary word
      Until no more words in the dictionary
      RETURN MDICW = False
    End if
  End
```

D.9 Logical Function NXTWRD

Purpose

NXTWRD finds the next word in BUFFER and deletes all delimiting characters between the current position in BUFFER and the beginning of the next word. If there is no other word in BUFFER, NXTWRD returns FALSE. Otherwise it returns TRUE.

Arguments: BUFFER, BEGWRD (see Appendix C)

Algorithm:

```
  Begin
    If no non-delimiting character in BUFFER then
      NXTWRD = FALSE
    Else
      Clear BUFFER from BEGWRD to the first non-delimiting character
      NXTWRD = TRUE
    End if
    RETURN
  End
```

D.10 Logical Function PRMPT

Purpose

PRMPT prompts the user for input and reads input into BUFFER starting at the *next buffer position* (which is at BUFPOS + 1). It returns TRUE if successful, FALSE otherwise. PRMPT calls subroutine DELETE to process any *Delete Letter* or *Delete Word* characters. This may cause deletion of previous input. If this occurs, the current dictionary position, stored in DICPOS, is reset to 1.

Arguments: BUFFER, BUFPOS, PROMPT, BEGWRD, DICPOS (see Appendix C)

Algorithm:

```
  Begin
    Repeat
      If (END-OF-BUFFER) then
        PRMPT = FALSE
      Else
```

```
      PRMPT = TRUE
      WRITE prompt to user
      READ response into BUFFER starting at BUFPOS+1
      Process any Delete characters (i.e. Delete Letter or Delete Word)
      If characters before BUFPOS have been deleted then DICPOS = 1
    End
    Until BUFFER beyond BEGWRD contains input
  End
```

D.11 Subroutine UPCASE

Purpose

UPCASE converts letters from lower case to upper case.

Argument: CHAR1

where

CHAR1 = A character string of length 1

Algorithm:

```
  Begin
    If the character is a lower case letter of the alphabet then
      Use the implicit FORTRAN function ICHAR to convert CHAR1 into
      its ASCII decimal equivalent (ADE)
      Subtract (ICCHAR('a')-ICCHAR('A')) from the ADE in order to obtain
      the ADE of the upper case letter
      Use the implicit FORTRAN function CHAR to convert the ADE into
      the upper case letter
    End If
    Return CHAR1
  End
```

D.12 Logical Function WRDTRM

Purpose

WRDTRM takes appropriate action if the character in the buffer is the *Help* character, the *Completion character* or a *delimiter* as defined in the function DLIMIT.

Arguments: BUFFER, NWRDS, DICT, BEGWRD, BUFPOS, DICPOS, HLPMSG
(see Appendix C)

Internal Variables:

DICCHR = Pointer to characters in the current dictionary word.

IDICP = Variable used to iterate over dictionary position so that
DICPOS need not be altered.

TRMCHR = The termination character initially at BUFPOS in BUFFER

LSTCHR = The number of characters in the current word not
including the termination character

Algorithm:

Begin

Initialize TRMCHR and LSTCHR

If the TRMCHR is a *Delimiting character* then

If LSTCHR = the length of the dictionary words OR if the current
BUFFER word + the *Padding character* matches a dictionary word
then RETURN WRDTRM = TRUE

End if

Find all the dictionary words which match the current word

If TRMCHR = *Help character* then

WRITE a help message followed by all the matching words
Set BUFPOS to the character preceding TRMCHR
WRDTRM = FALSE

Else if there is a unique matching dictionary word

If TRMCHR = *Completion character* then

WRDTRM = FALSE

Complete the word in the buffer

Else

WRDTRM = TRUE

End if

Else

Write an error message

If TRMCHR = *Completion character* then complete word as far as
possible

End if

End

Appendix E

MACHINE DEPENDENCIES

The following is a list of the implementation decisions which are machine dependent. To the authors knowledge these are the only non-ANSI features of the GETWRD Package.

- 1) In several places use is made of characters not contained in the standard ANSI character set. However, all characters are contained in the ASCII character set. Non-ANSI characters are used in the following places:
 - a) The default characters "^", "\", "<", and "?" for the special characters: the *Abort character*, the *Delete Letter character*, the *Delete Word character*, and the *Help character*.
 - b) The lower case error messages output by the procedures DICCHK, GETNUM, GETWRD, PROMPT, and WRDTRM.
 - c) The subroutine UPCASE which is used to convert from lower to upper case letters when matching dictionary words.
 - d) The lower case comments throughout the GETWRD Package.
- 2) In the procedure PROMPT, the first WRITE statements contain non-ANSI FORMAT specifiers.

```
WRITE(UNTOUT, '( /1X,2A$') ) PROMPT, BUFFER(:BUFPOS)  
WRITE(UNTOUT, '( /1X,A$') ) PROMPT
```

The "\$" sign at the end of each FORMAT specifier, is used by DEC-20 FORTRAN 7.0 to suppress the carriage return. The characters which are then READ from the terminal (the usual input device), into the character variable BUFFER, appear on the terminal screen on the terminal screen on the same line as the current contents of BUFFER. Deletion of the "\$" characters will cause the input characters to appear on the following line.

GETWRD Package

Subject Index

- ^, default Abort character 4, 19
- \$ or <dollar sign>, default Completion character 3, 15, 18
- <, Default Delete Word character 4, 18, 19
- <Blank>, default Delimiting character 2
- <Hat sign>, default Abort character 4, 19
- <Space>, default Delimiting character 2
- ?, Default Help character 2, 19
- ABORT 19, 24, 27, 29
- Abort character 4, 7, 19, 25
- BEGWRD 23, 29, 30, 31
- Buffer 4, 8, 9, 10, 11, 13, 22, 23, 26, 28, 29, 30, 31
- Buffer initialization 2, 6, 9, 10, 23
- Buffer overflow 6, 8, 9, 10, 11, 26, 27, 28
- Buffer pointer 6, 9, 10, 22
- Buffer, deletion of characters in buffer 4, 23
- Buffer, purpose of 2, 30
- BUFPOS 9, 10, 13, 22, 23, 26, 28, 29, 30, 31
- CLBUFF 6, 13, 23
- CMPLT 19, 24, 27, 29
- Completion character 3, 7, 19, 25
- Completion character, processing of 31
- DELETE 23-24
- Delete Letter character 4, 7, 19, 23, 25
- Delete Word character 4, 7, 19, 23, 25
- Delimiter, processing of 31
- Delimiting characters 2, 6, 7, 19, 22, 24, 25, 26, 30
- Delimiting characters, definition of 26
- DELLTR 19, 24, 27, 29
- DELWRD 19, 24, 27, 29
- DICCHK 6-8, 11, 24-26, 33
- DICPOS 13, 28, 29, 30, 31
- DICT 9, 11, 13, 22, 24, 28, 29, 31
- Dictionary 2, 4, 8, 10, 22, 28
- Dictionary, checking of 6-8, 11
- Dictionary, in upper case 22
- Dictionary, purpose of 2
- Dictionary, sorted entries 22, 29
- Dictionary, use in matching user's response 29
- Dictionary, use in printing a help message 2
- Dictionary, use in word completion 3
- DLIMIT 19, 26, 31
- DLMTR 11
- Environment of the GETWRD Package 21
- Friendliness, in a program/user interface 1
- GETNUM 2, 6, 10-11, 14, 15, 26-28, 33
- GETWRD 2, 6, 8-10, 11, 13, 28, 33
- Help character 2, 7, 19, 25
- Help character, processing of 31
- Help message 2, 8, 10, 22
- HIBUFP 23
- HLPCHR 19, 24, 27, 29
- HLPMSG 9, 10, 13, 22, 26, 28, 31
- I/O device, modification of default 19
- IER 10, 26
- Implementation of a Command Language 1
- INSBUF 28-29
- LOBUFP 23
- Matching dictionary words 2
- MDICW 22, 29-30
- NCHR 28
- Number, inputting a real number 26

GETWRD Package

NWRDS 9, 12, 13, 22, 24, 28, 29, 31
NXTWRD 6, 11, 13, 30

PADCHR 12
Padding character 6, 22, 23, 25
PRMPT 30-31
Prompt 1, 8, 9, 10, 13, 22, 26, 28, 30, 33
Prompt, getting help at a prompt 2
Prompt, suppression of 2

RNUM 26

Special characters, modification of
defaults 18
Spelling correction 1, 10

Type-ahead 1, 2, 11

UPCASE 31

Word completion 1, 3-4, 10, 28
Word matching 2
Word, definition of 2
Word, getting the next word 28
WRDTRM 31-33

\, Default Delete Letter character 4, 18,
19

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D		
(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)		
1. ORIGINATING ACTIVITY	2a. DOCUMENT SECURITY CLASSIFICATION Unclassified	
	2b. GROUP	
3. DOCUMENT TITLE GETWRD PACKAGE		
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) Technical Memorandum		
5. AUTHOR(S) (Last name, first name, middle initial) Hally, David; Dent, C. Ann		
6. DOCUMENT DATE March 1984	7a. TOTAL NO. OF PAGES 42	7b. NO. OF REFS 0
8a. PROJECT OR GRANT NO.	9a. ORIGINATOR'S DOCUMENT NUMBER(S) DREA Technical Memorandum 84/D	
8b. CONTRACT NO.	9b. OTHER DOCUMENT NO.(S) (Any other numbers that may be assigned this document)	
10. DISTRIBUTION STATEMENT		
11. SUPPLEMENTARY NOTES	12. SPONSORING ACTIVITY	
13. ABSTRACT The GETWRD Package is a library of twelve procedures designed to ease the implementation of command languages in FORTRAN programs. Use of the GETWRD Package saves the implementor of the program from designing and coding new input routines. The GETWRD Package is able to interpret a word of user input by matching it with one of the entries in a user-supplied dictionary. Alternatively the word may be interpreted as a number. The GETWRD Package includes many features designed to increase the friendliness of the program/user interface. These include a type ahead facility, recognition of individual tailoring of prompts and help messages and, where possible, help messages include the list of allowed responses. A procedure which checks the dictionary for errors is also supplied as a debugging tool. In this memorandum a complete description of each procedure is given, including the algorithm used, discussion of their use and examples. The GETWRD Package is written in ANSI standard FORTRAN of 1977. Care has been taken to make the package as portable as possible.		

17518
7a-070

KEY WORDS

Computer/user interface
Command languages

INSTRUCTIONS

1. **ORIGINATING ACTIVITY:** Enter the name and address of the organization issuing the document.
- 2a. **DOCUMENT SECURITY CLASSIFICATION:** Enter the overall security classification of the document including special warning terms whenever applicable.
- 2b. **GROUP:** Enter security reclassification group number. The three groups are defined in Appendix M of the DRS Security Regulations.
3. **DOCUMENT TITLE:** Enter the complete document title in all capital letters. Titles in all cases should be unclassified. If a sufficiently descriptive title cannot be selected without classification, show title classification with the usual one-capital-letter abbreviation in parentheses immediately following the title.
4. **DESCRIPTIVE NOTES:** Enter the category of document, e.g. technical report, technical note or technical letter. If appropriate, enter the type of document, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.
5. **AUTHOR(S):** Enter the name(s) of author(s) as shown on or in the document. Enter last name, first name, middle initial. If military, show rank. The name of the principal author is an absolute minimum requirement.
6. **DOCUMENT DATE:** Enter the date (month, year) of Establishment approval for publication of the document.
- 7a. **TOTAL NUMBER OF PAGES:** The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.
- 7b. **NUMBER OF REFERENCES:** Enter the total number of references cited in the document.
- 8a. **PROJECT OR GRANT NUMBER:** If appropriate, enter the applicable research and development project or grant number under which the document was written.
- 8b. **CONTRACT NUMBER:** If appropriate, enter the applicable number under which the document was written.
- 9a. **ORIGINATOR'S DOCUMENT NUMBER(S):** Enter the official document number by which the document will be identified and controlled by the originating activity. This number must be unique to this document.
- 9b. **OTHER DOCUMENT NUMBER(S):** If the document has been assigned any other document numbers (either by the originator or by the sponsor), also enter this number(s).
10. **DISTRIBUTION STATEMENT:** Enter any limitations on further dissemination of the document, other than those imposed by security classification, using standard statements such as:
 - (1) "Qualified requesters may obtain copies of this document from their defense documentation center."
 - (2) "Announcement and dissemination of this document is not authorized without prior approval from originating activity."
11. **SUPPLEMENTARY NOTES:** Use for additional explanatory notes.
12. **SPONSORING ACTIVITY:** Enter the name of the departmental project office or laboratory sponsoring the research and development. Include address.
13. **ABSTRACT:** Enter an abstract giving a brief and factual summary of the document, even though it may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall end with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (TS), (S), (C), (R), or (U).

The length of the abstract should be limited to 25 single-spaced standard typewritten lines; 7 1/2 inches long.
14. **KEY WORDS:** Key words are technically meaningful terms or short phrases that characterize a document and could be helpful in cataloging the document. Key words should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context.



**D
R
E
A**



**C
R
D
A**